Optimizing internal logistic flows in hospitals by dynamic pick-up and delivery models

Wim Vancroonenburg · Eline Esprit · Pieter Smet · Greet Vanden Berghe

Abstract In the present contribution, the application of a dynamic pick-up and delivery model for hospital internal logistics is investigated. Two scheduling policies are developed: the first applies a *cheapest insertion* heuristic targeting optimization of the problem's objective function, while the second employs local search to improve the current schedule. A computational study is presented in which the two policies are applied on a range of generated problem instances that have been made available to the research community. The benefit of the two policies is demonstrated by comparing them with a common and intuitive *earliest first, by due date* policy. Secondly, the benefit of allowing to combine transports is also investigated, and it is shown that this may lead to further increased performance.

Keywords Hospital logistics · Patient transportation · Dynamic pick-up and delivery · Local search

1 Introduction

Internal logistics play a fundamental role in the daily operation of any hospital. It is the backbone service on which most hospital activities depend. Several internal logistic flows are distinguishable: the supply of clean linen, food and medicine; correspondingly the retrieval of waste, dirty linen, food trays and used surgical instruments; and the transportation of patients from/to consultation, medical imaging, and surgeries. The diversity of these logistic flows, each with its own distinct rules and complexities, makes managing this process a daunting task. To address this situation, we present a model for organizing

W. Vancroonenburg (corresponding author), E. Esprit, P. Smet, G. Vanden Berghe
KU Leuven, Technology Campus Ghent
Department of Computer Science, CODeS & iMinds-ITEC
Gebroeders De Smetstraat 1, 9000 Gent, Belgium
Tel.: +32 9 265 87 04
E-mail: wim.vancroonenburg@kuleuven.be

and directing these logistic flows based on the dynamic pick-up and delivery problem (DPDP). DPDPs are one class of vehicle routing problems (VRPs) which focus on goods requiring pick-up from and delivery to specific locations by a fleet of vehicles (see Berbeglia et al. [2] for a survey). The organization of logistic transports in hospitals fits naturally within this problem class. Moreover, *soft* aspects imposed by hospital policies and management make it a challenging variant. For example, certain routes or corridors in a hospital may be excluded for patient transports, while they are allowed for goods transportation. Thus, transport time is dependent upon what exactly is being transported. A further example of a *soft* aspect is when multiple goods may be transported together, whereas patient transportation is ideally not combined with other transports. The dynamic aspect of the problem is also of specific interest, as it is characterized by relatively high request arrival rates and short transport times due to the short distances between pick-up/deliveries in a hospital. Therefore, dispatching decisions must be made in a relatively short amount of time.

Although the application of operations research techniques to support dynamic (real-time) dispatching for logistics is not new, with contributions as early as the 1980's [7], research interest in this topic has only been expanding rapidly in the past decade [8]. As also noted by Berbeglia et al. [2], this generally derives from technological advances such as faster computing systems and innovative algorithms in addition to new real-time communication technologies (such as smartphones) which are necessary for the practical implementation of such systems. Given the limited amount of time in which decisions need to be made (e.g. when vehicles need to be directed to their next destination, or when new requests are announced), a clever use of the time between events may allow for the use of inventive approaches that achieve improved performance over common dispatching rules. An interesting, early contribution in that respect is presented by Gendreau et al. [4], in which neighbourhood search using ejection-chains [5] is applied to the DPDP between the arrival of events.

In health care, only few such dynamic systems have been developed, mostly for patient transport due to its strong ad-hoc nature. However, dispatching is often still done manually, or using simple dispatching rules-of-thumb that certainly leave room for improvement. Recent studies have looked at more advanced methods to solve DPDP problems for patient transports. Beaudry et al. [1] discuss the application of the dynamic dial-a-ride problem (DDARP), a variant of the DPDP, for organizing intra hospital patient transport in a German hospital. Their study mostly focuses on transports between buildings in a 100+ buildings hospital site. This application ultimately resulted in the development of the Opti-TRANS system [6]. Fiegl and Pontow [3] discuss the development of an algorithm which dynamically schedules general pick-up and delivery tasks (patients, but also lab results, materials and other such items) within hospitals. Their approach is strongly focussed on minimizing the average weighted flow time, enabling high task throughput. It is however less suitable for a general objective function, quite possibly featuring several (possibly conflicting) objectives.

The scope of this paper is restricted to the transportation of patients within a hospital. The combination with material logistic flows, that also involve regular transports planned in a logistic (typically weekly) timetable, will be investigated in a subsequent study. Nevertheless, the present contribution is also applicable to ad-hoc material transports that may be combined with patient transports, though the focus will be on the latter.

The main research question for this study is to investigate the efficiency of scheduling policies that specifically target optimization of hospital KPI's, formulated through a weighted sum objective function. A definition for the hospital logistics problem is presented that includes elements common to DPDP definitions combined with elements relevant to the hospital context. A computational study is performed using two scheduling policies which are applied to a range of randomly generated, realistic instances, that have been made available to the research community. An important question that is also investigated is whether it is beneficial to combine patient transports, that is, transport multiple patients together. It can be envisioned that it is beneficial to e.g. combine two patient transports when they share a route (partially). This aspect is also investigated in the computational study.

2 Problem statement

The primary elements and attributes associated with the dynamic hospital logistics scheduling problem studied in this paper are:

Location: The hospital and its layout can be represented by an undirected graph $G(V, A, C)$ where $V = \{v_1, \ldots, v_n\}$ is the set of vertices representing all relevant locations, such as rooms, doors, junctions in corridors. $A = \{\{v_i, v_j\} | \{v_i, v_j\} \in V \times V, i \neq j\}$ is the set of arcs connecting different locations and junctions, thus representing corridors, stairs and elevators. $C : A \times 2^R \rightarrow \mathbb{R}$ is a travel time function, mapping an arc and a transport request (see further) subset to the travel time along that edge, carrying the current request subset. Hence, the travel time between two locations may differ due to the current set of requests being executed together (e.g. transporting both a walking patient and a patient by wheelchair may limit the walking speed).

Request: A request represents the order for a transport that must be performed and is denoted by $r = (r^p, r^d)$, with R representing the set of all requests. It consists of a pick-up task r^p at location $v_r^p \in V$ and a delivery task r^d at location $v_r^d \in V$. A request is announced at a certain time a_r. A time-window $[st^{p/d}, dt^{p/d})_r$ is specified for both the pick-up (denoted with superscript p) and the delivery (denoted with superscript d) in which the pick-up/delivery should occur. Finally, a service time $s_r^{p/d}$ must also be considered when performing a pick-up/delivery.

Porter: Porters represent the staff available for executing pick-up and delivery tasks. Porters have an availability time-window during which they may be assigned tasks. They start and stop their day task at a certain depot $v_k \in V$. Furthermore, it is assumed that they take no breaks, however clearly in a practical application this must be accounted for.

Skill: Requests may require a certain skill of the porter performing the transport. For example, patient transports by wheelchair are normally only executed by qualified care staff, while material transports can be executed by both regular logistic staff and care staff. Therefore, we assume porters may have one or more skills (may transport patients, may transport medicine), and requests may be constrained by a skill dependency.

Forbidden combination: A forbidden combination is a set of requests $\{r, \ldots, r'\}$ that may not be performed simultaneously by one porter (e.g. transporting a patient while carrying dirty linen).

The problem involves assigning and scheduling the transport requests to porters, respecting forbidden combinations and required porter skills. However, no strict policy is imposed as to when requests should be assigned/scheduled – a request may be queued and assigned at a later time. For example, it is possible to queue the requests and schedule the most urgent request when a porter has completed a transport. Time-windows of requests are considered semi-soft: a request pick-up/delivery may not be scheduled before the start of its corresponding time-window, however it can be scheduled after the end of this time-window. All requests should be scheduled, and a weighted objective function of several KPIs should be minimized:

1. Total tardiness: the tardiness of a request pick-up / delivery is defined as: $T_r^{p/d} = Max(0, C_r^{p/d} - dt_r^{p/d})$ with $C_r^{p/d}$ the completion time of the pick-up/delivery of request r.
2. Porter overtime: the overtime of a porter is defined as the completion time of the last request performed by the porter minus the end of his availability time-window (or 0, if negative).
3. Total travel time: the travel time of a porter is defined as the time spent on moving towards pick-up/delivery locations, plus the final return to their depot.

In the present context of hospital logistics, the KPIs correspond with, for example, minimizing the lateness of consultation appointments (where d_r^d equals the assigned consultation time) or minimizing post-consultation waiting times (where $st_r^p = dt_r^p$ and equals the end of the patients' consultation). The definition of time-windows is flexible, thus enabling modelling of multiple aspects. In the remainder of this study, the problem is denoted as the DPDP-Hosp: the dynamic pickup and delivery problem in hospitals.

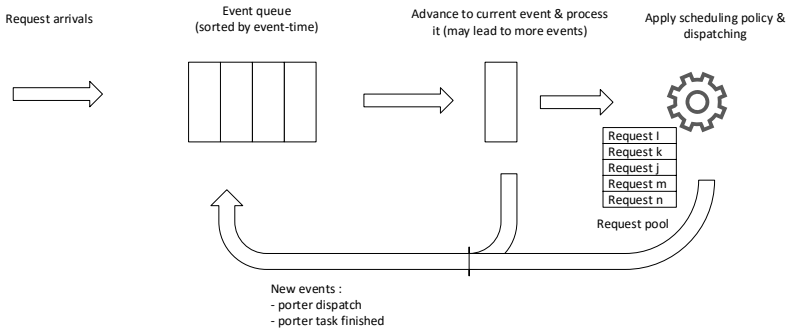

Figure 1 Visual representation of the event-based framework. The main source of events are *request arrivals*. The discrete event environment will advance at each iteration to the next event time in the queue, process the current event (a porter dispatch will lead to updating the current state of the simulation, by assigning the task to the porter; it also queues the porter task finish in the future) and then allow a scheduling policy to perform any operation on the current state. This includes changing the scheduled tasks on the current staff, scheduling new tasks from the request pool and posting dispatch events.

3 Solution approach

3.1 Framework

The DPDP-Hosp is modelled and solved in a discrete event-based framework. The main source of events is the *request arrival*, i.e. the announcement of a request r at its announce-time a_r (in which all its information, pick-up/delivery task, location, time-window, is made available). Additionally, two event types are considered:

- Porter dispatch: this event is announced when a porter departs towards a pick-up/delivery location;
- Porter service end: this event is announced when a porter finishes a pick-up/delivery service at the respective location.

Algorithm 1 describes the main flow of this event-based framework, also depicted in Figure 1. In essence, the framework processes events (denoted by e) in order of their announce time ($time(e)$) and allows a *scheduling policy* ($SchedulePolicy$) to handle the current event, update the current state, remove requests from the request queue (request pool Rp), etc.

The main data structure of importance, not explained in the pseudo-code, is the current *state* (denoted S). Next to maintaining where each porter currently is and what requests have been executed, it maintains two important aspects:

- which task each porter is currently executing (if any), and
- which tasks have been assigned and queued to each porter, along with their scheduled start time (start of service, not dispatch) and end time.

Algorithm 1 Discrete event-based framework pseudocode

Require: $G(V, E, C)$
Require: P
Require: $R = (r_1, r_2, \ldots, r_N)$
  $t \leftarrow 0$ ▷ Simulation time t is initially 0
  $eq \leftarrow (r_1, r_2, \ldots, r_N)$ ▷ Event-queue eq is initialized with all request arrivals
  $Rp \leftarrow \emptyset$ ▷ Request pool of announced events is initially empty
  $S \leftarrow InitializeCurrentState()$
  while eq not empty **do**
    $e \leftarrow popFront(eq)$ ▷ Get next event
    $t \leftarrow time(e)$ ▷ Advance time
    if e is request arrival **then**
      $Rp \leftarrow Rp \cup \{e\}$ ▷ Add request to request pool
    end if
    if e is porter dispatch **then**
      $p \leftarrow porter(e)$ ▷ Get porter
      $r^{p/d} \leftarrow task(e)$ ▷ Get assigned task (pick-up/delivery).
      $v \leftarrow loc(e)$ ▷ Get location to where porter is dispatched
      $S \leftarrow UpdateCurrentState()$
      $d(loc(p), v) \leftarrow travelTime(G, S, p, v)$ ▷ Determine travel time to v
      $e' \leftarrow porterServiceEnd(t + d(loc(p), v) + s_r^{p/d}, p, v)$ ▷ Create service end event
      $Insert(e', eq)$ ▷ Add event to event queue
    end if
    if e is porter service end **then**
      $S \leftarrow UpdateCurrentState()$
      $p \leftarrow porter(e)$ ▷ Get porter
      $v \leftarrow loc(e)$ ▷ Get location where porter finished service
      $r^{p/d} \leftarrow nextTask(S, p)$ ▷ Get next assigned task for porter
      if $r^{p/d}$ exists **then**
        $e' \leftarrow porterDispatch(Max(st_r^{p/d} - d(v, v_r^{p/r}, t), p, v_r^{p/r})$
        ▷ Create new porter dispatch event, if any
      end if
    end if
    $SchedulePolicy(t, e, eq, S, Rp)$
  end while

This task assignment to porters is visualized in Figure 2.

3.2 Scheduling policies

Two policies are presented that aim at minimizing the KPI's presented in the previous section: a cheapest insertion heuristic, and a local-search method using the former insertion heuristic in an iterative manner.

3.2.1 Cheapest insertion

Given the current state at the announce time of a new request, this policy will insert the request at the lowest possible cost. The policy will determine a position in the scheduled tasks of a porter such that the cost of inserting the request is as small as possible.

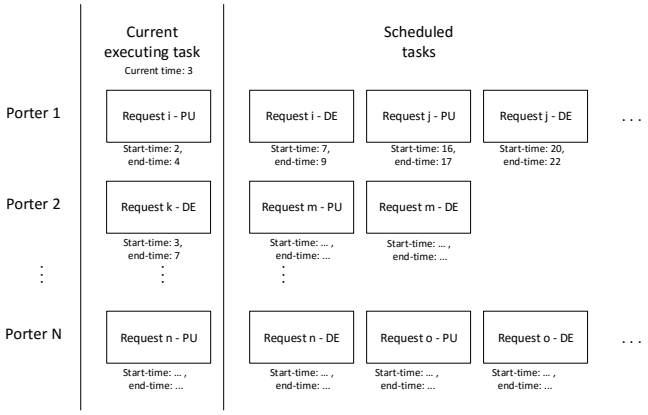

Figure 2 Visual representation of the current state at any given time t. Porters may be executing a task, and have a list of currently queued tasks scheduled and assigned to them. These queued tasks are *volatile*: policies may decide to reorder/redistribute these tasks, as long as they are not dispatched (i.e. currently executing).

Two variants of the cheapest insertion heuristic are possible, depending on whether or not the request delivery must be scheduled immediately after the pick-up. The implementation can handle both cases, using a parameter to indicate if consecutive pick-ups and deliveries are required.

Certain constraints must be taken into account to ensure feasibility. Firstly, tasks requiring special skills may only be assigned to porters having the appropriate competences. Hence, only porters having the required skills are considered when evaluating positions for inserting the request. Furthermore, in the case of non-consecutive pick-ups and deliveries, if the request types of two tasks form a forbidden combination, those tasks may never be executed simultaneously. A porter may not pick up a request before having delivered all other requests forming a forbidden combination with it. For each non-consecutive set of positions for the request's pick-up and delivery, the other tasks of the porter must be checked for forbidden combinations.

After determining a feasible insertion position in a porter's list of scheduled tasks, the start and end times of the tasks are recalculated. This is done using a simple procedure, iterating over all scheduled tasks and determining the start time of a task[1] based on the maximum value of:

- the current decision time t, incremented with the travel time to the pick-up/delivery location,
- the start of the time-window $st_d^{r/p}$.

[1] Note that this is the start time of the pick-up/delivery service, not the dispatch time to the location;

3.2.2 Local search

The second policy is an extension of the previous policy. After inserting a new request in the solution using the method described previously, a simple local search-based procedure is applied to the current state. At each iteration, a request is randomly selected and ejected from the porter's route. Using the cheapest insertion procedure, the request is inserted into the schedule again and the start and end times of the scheduled tasks are recalculated. New solutions are accepted if their cost is lower than or equal to the best known cost. The search is stopped if no improvements are found for a certain number of iterations. Currently, it is assumed that there is sufficient time between the arrival of two requests to allow the search process to finish. In the computational study following below, this is generally the case.

4 Computational study

4.1 Experimental setup

The computational study is based on a real-life hospital layout of the newly constructed AZ Maria Middelares, a 7-level (+1 subterranean level) hospital in Ghent, Belgium. A detailed graph was mapped on the hospital layout, with accurate alignment to latitude/longitude coordinates. All rooms, corridors, stairs and elevators were included in the graph as vertices and edges.

The discrete event-based framework was implemented as a simulation environment, developed in Java 1.8 (and shown in Figure 3), in which different scheduling and dispatching policies can be tested on both generated and historic data.

The computational study focuses on the comparison of the cheapest insertion-policy (denoted **CI**) with the local search-policy (denoted **LS**). A comparison is also made with a common dispatching rule, *earliest first, by due date* (denoted **EFDD**). The latter policy is seemingly intuitive, processing the tasks in a *first due* order and assigning them to the porter who can start the service earliest (based on porters' last assigned task's planned end time, and the travel time to the current task). An initial comparison is made between the three policies EFDD, CI and LS on the KPI's defined in Section 2. To this end, the executed schedules are each evaluated at the end of a simulation run. The three objectives, total tardiness, porter overtime and total travel time, are combined as terms in a linear weighted sum objective function, with weights representing the relative importance of the objectives to the decision maker. For the purpose of this study, these weights have been set to one; i.e. the three objective terms are simply summed in the final objective value.

Additionally, the benefit of allowing to combine transports, respecting forbidden combinations is investigated both for CI and LS. To this end, we allow to combine up to three transport requests in the CI and LS heuristics.

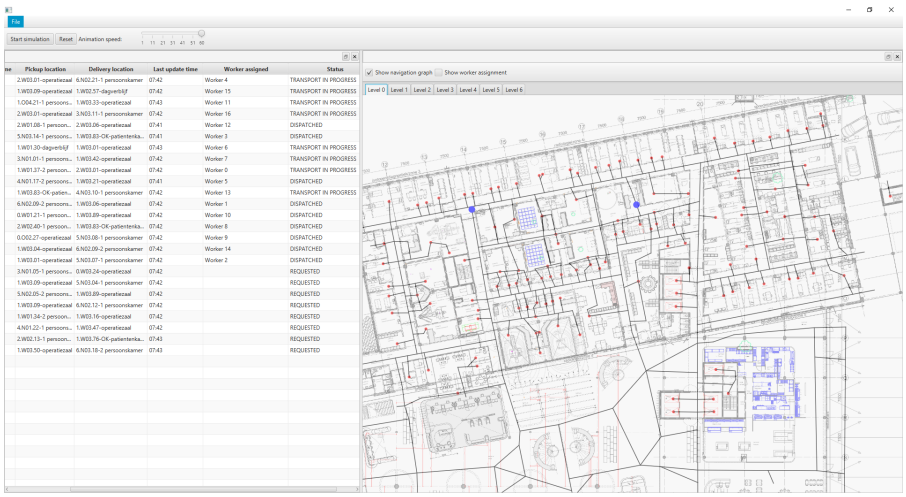

Figure 3 Discrete event simulation environment developed in Java 1.8, showing (partially) one level of the hospital layout with overlaid graph.

The policies are tested on randomly generated request arrival scenarios having the following characteristics:

- Pick-up/delivery locations are randomly selected between two groups of hospital rooms: consultation/exam/operating rooms and patient rooms and waiting areas. Hence, the transport pairs largely correspond to the bulk of patient transports from their room to an appointment in the operating theatre or to a physician.
- Service times are exponentially distributed with mean 30 seconds (i.e. it is assumed the pick-up/delivery is normally rather fast)
- Time-windows of 15 minutes, with the end of the time-window, relative to the request announce time, log-normally distributed. Either the pick-up or the delivery time-window is *binding* (the other being very large), with a 50/50 ratio between the two cases.
- A homogeneous Poisson-process (i.e. exponential inter-arrival) with rate parameter λ. Requests are of 3 types: by wheelchair, by bed, or walking. Only walking patients may be grouped in a single transport.
- A workforce of N porters.

A dataset of instances with different arrival rates ($\lambda = 0.1, 0.5, 1.0$ arrivals/minute) and size of workforce ($N = 5, 10, 15$) was created, with 10 randomly generated instances per combination. These instances are provided in an accessible JSON-format[2].

[2] Available at `https://gent.cs.kuleuven.be/hospital-logistics/`

4.2 Results and discussion

4.2.1 No combined transports (consecutive pick-up and deliveries)

Firstly, the performance of the CI and LS policies is compared to the baseline policy EFDD, without allowing combining transports. Table 1 reports the lateness, overtime, travel time and combined objective value for each policy, averaged over ten instances of each combination of λ and N. It is clear that CI and LS outperform the baseline policy on all cases. Both policies have a clear emphasis on minimizing the travel time, which has the strongest impact on the objective value. Hence, it is beneficial to employ a scheduling policy that specifically targets optimization of the weighted objective value, as defined. Though the sensitivity of this experiment to these weights remains an open question, similar results can be expected where the CI and LS policies will target the largest impacting KPI.

4.2.2 Combining transports

The second research question considers whether or not it is beneficial to combine transports. Table 2 reports only on the objective values obtained for each policy, again averaged over ten instances of each combination of λ and N. The table includes CI and LS configurations that combine transports, allowing up to three (parameter Cap) transports to be combined. The results show that combining transports is certainly beneficial, for both CI and LS policies. Allowing more than two transports is mostly beneficial, but this is not always the case.

λ	N	EFDD				CI (consec.)				LS (consec.)			
		Late.	Over.	Travel.	Obj.	Late.	Over.	Travel.	Obj.	Late.	Over.	Travel.	Obj.
0.1	5	50.57	11.98	198.21	260.76	3.05	13.15	181.26	197.46	2.40	13.11	179.22	**194.73**
0.1	10	23.50	2.07	190.69	216.27	1.47	1.85	170.11	173.44	1.25	1.94	167.73	**170.92**
0.1	15	71.14	29.27	194.07	294.49	1.56	30.74	172.80	205.09	1.35	30.74	170.89	**202.98**
0.5	5	263.10	45.96	988.65	1297.71	138.25	69.23	846.20	1053.67	116.07	68.76	812.36	**997.18**
0.5	10	237.85	46.23	981.49	1265.57	45.34	66.04	839.77	951.15	34.50	65.98	799.92	**900.40**
0.5	15	227.53	41.87	992.93	1262.33	19.10	60.58	829.75	909.43	10.92	60.21	794.69	**865.82**
1	5	588.22	103.76	1960.83	2652.80	581.48	157.23	1655.85	2394.56	438.85	153.86	1556.87	**2149.58**
1	10	561.78	72.02	1969.16	2602.96	279.87	161.65	1622.66	2064.18	221.40	159.16	1535.26	**1915.82**
1	15	565.35	85.36	1996.42	2647.13	143.74	176.72	1639.98	1960.45	102.68	175.91	1544.94	**1823.53**

Table 1 Overview of the performance (lateness, overtime, travel time and combined objective value) for each policy, averaged over 10 instances of each combination of λ and N. For CI and LS, the table does not include results for allowing combined transports (i.e. pick-up and delivery are consecutive). Bold highlighting indicates best result.

λ	N	EFDD	CI			LS		
			consec.	non-consec.		consec.	non-consec.	
				$Cap = 2$	$Cap = 3$		$Cap = 2$	$Cap = 3$
0.1	5	260.76	197.46	196.46	196.50	194.73	193.66	**192.66**
0.1	10	216.27	173.44	169.80	169.25	170.92	168.96	**168.12**
0.1	15	294.49	205.09	203.37	203.43	202.98	**201.56**	201.66
0.5	5	1297.71	1053.67	1029.91	1016.92	997.18	961.07	**943.81**
0.5	10	1265.57	951.15	921.01	908.00	900.40	868.78	**861.34**
0.5	15	1262.33	909.43	896.58	881.54	865.82	847.46	**833.12**
1	5	2652.80	2394.56	2282.92	2219.32	2149.58	2051.27	**2002.03**
1	10	2602.96	2064.18	1978.83	1932.45	1915.82	1815.13	**1776.50**
1	15	2647.13	1960.45	1873.51	1818.70	1823.53	1722.97	**1685.84**

Table 2 Overview of the average objective-values for each policy, on each combination of λ and N. Consec. denotes that the policy respects the consecutiveness of pick-up and deliveries, i.e. no transports are combined. Non-consec. allows combining transports, respecting forbidden combinations and a maximum of combined transports equal to Cap. Bold highlighting indicates best result.

5 Conclusion

The past decade has seen a strong increase in the (successful) application of dynamic pickup and delivery models, and vehicle routing models in general, to support (real-time) decision making in logistics. Hospitals, until now, have been lagging behind, with internal logistic processes mostly being scheduled and dispatched manually.

Only recently are systems being deployed to automate this process by means of dispatching rules. This paper showed that such systems, and thus hospitals, can benefit by applying more advanced scheduling policies that specifically target optimization of hospitals' KPIs. Two such policies were presented: a first policy that uses a cheapest insertion (in terms of a weighted sum objective function) heuristic to insert transport tasks in porters' schedules, and a second policy that improves on that by applying local search. Though relatively basic, these two policies already show improved performance in comparison to an intuitive dispatching rule, *earliest first, by due date*.

An interesting aspect of applying DPD models to internal hospital logistics, is the possibility of combining transports. Whereas in e.g. DPD applications for parcel delivery services it may be common practice to combine transports (multiple pick-ups in sequence, before performing deliveries), for patient transports in hospitals this is certainly not the case. Nevertheless, the possibility of combining transports exists and this paper shows it is also beneficial to allow this, even on only a limited portion of transports. Hence, this study may convince hospital managers to reconsider this possibility, in light of improved performance.

Acknowledgements Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX. (`http://comex.ulb.ac.be`) and funded by the iMinds ICON project 'AORTA' (`http://www.iminds.be/nl/projecten/2015/03/10/aorta`).

References

1. Beaudry, A., Laporte, G., Melo, T., Nickel, S.: Dynamic transportation of patients in hospitals. OR Spectrum **32**(1), 77–107 (2008)
2. Berbeglia, G., Cordeau, J.F., Laporte, G.: Dynamic pickup and delivery problems. European Journal of Operational Research **202**(1), 8–15 (2010)
3. Fiegl, C., Pontow, C.: Online scheduling of pick-up and delivery tasks in hospitals. Journal of biomedical informatics **42**(4), 624–32 (2009)
4. Gendreau, M., Guertin, F., Potvin, J.Y., Séguin, R.: Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Transportation Research Part C: Emerging Technologies **14**(3), 157–174 (2006)
5. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics **65**(1), 223 – 253 (1996)
6. Hanne, T., Melo, T., Nickel, S.: Bringing Robustness to Patient Flow Management Through Optimized Patient Transports in Hospitals. Interfaces **39**(3), 241–255 (2009)
7. Psaraftis, H.N.: A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. Transportation Science **14**(2), 130–154 (1980)
8. Psaraftis, H.N., Wen, M., Kontovas, C.A.: Dynamic vehicle routing problems: Three decades and counting. Networks **67**(1), 3–31 (2016)